1. ¿Qué es *Transfer Learning*?

Transfer Learning (Aprendizaje por Transferencia) es una técnica donde un modelo preentrenado en una tarea se reutiliza para una nueva tarea relacionada. La idea es que el conocimiento aprendido en la primera tarea sea útil para acelerar y mejorar el aprendizaje en la segunda tarea.

Ejemplo práctico:

- Usar un modelo como **BERT** preentrenado en grandes corpus para una tarea específica como clasificación de sentimientos o reconocimiento de entidades (NER).

---

2. ¿Por qué son importantes los *embeddings*?

1. **Representación densa:**
    - Transforman palabras en vectores de dimensión fija, permitiendo a las máquinas trabajar con datos de lenguaje natural de manera eficiente.
2. **Captura de semántica:**
    - Los *embeddings* pueden capturar el significado de palabras y sus relaciones. Por ejemplo, palabras como *“gato”* y *“perro”* tendrán representaciones similares.
3. **Generalización:**
    - Mejoran el rendimiento de modelos al permitir que aprendan patrones semánticos complejos.
4. **Reutilización:**
    - Se pueden reutilizar *embeddings* preentrenados (como *Word2Vec* o *GloVe*) para acelerar el entrenamiento en tareas específicas.

---

3. ¿Qué es un *embedding*?

Un **embedding** es una representación vectorial densa de una palabra, frase o elemento discreto en un espacio continuo de baja dimensión.

Ejemplo:
La palabra *"gato"* podría representarse como el vector `[0.12, -0.34, 0.56, ...]`.

Los *embeddings* permiten que palabras con significados similares tengan vectores similares.

---

4. ¿Qué son *embeddings contextuales*?

Los **embeddings contextuales** generan representaciones de palabras que varían según el contexto en el que aparecen. A diferencia de los *embeddings* estáticos como *Word2Vec*, los contextuales consideran el significado de la palabra dentro de una oración específica.

Ejemplo:

- La palabra *"banco"* en:
  - *"Voy al banco a sacar dinero"* → Contexto financiero.
  - *"Me senté en el banco del parque"* → Contexto de asiento.

Modelos que producen embeddings contextuales:

- **BERT**, **ELMo**, **GPT**.

---

5. ¿Cómo se implementa el *Transfer Learning* en una red neuronal?

1. **Seleccionar un modelo preentrenado:**
   - Ejemplo: Usar **BERT** para tareas de NLP o **ResNet** para visión por computadora.
2. **Congelar capas iniciales:**
   - Se mantienen las capas preentrenadas congeladas (no se actualizan durante el entrenamiento).
3. **Añadir nuevas capas:**
   - Se agregan capas específicas para la tarea deseada (por ejemplo, una capa densa para clasificación).
4. **Entrenamiento en datos específicos:**

- Se ajusta el modelo a los datos de la nueva tarea (Fine-Tuning).

Ejemplo en código con PyTorch (NLP):

python

Copiar código

```
from transformers import BertForSequenceClassification


# Cargar BERT preentrenado
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)


# Entrenar (Fine-Tuning) el modelo en una tarea de clasificación específica
```

---

6. ¿Qué es *pre-training*?

El **pre-training** es la fase inicial en la que un modelo se entrena con grandes cantidades de datos no etiquetados para aprender representaciones generales del lenguaje o características visuales.

- En NLP, el preentrenamiento puede realizarse con tareas como **MLM (Masked Language Modeling)** o **Next Sentence Prediction**.
- En visión por computadora, se usan tareas como **clasificación de imágenes en ImageNet**.

---

7. Tengo un problema de clasificación, tengo los datos, ¿cómo lo resolverías?

1. **Preprocesamiento de datos:**
    - Tokenizar el texto si es NLP.
    - Normalizar características si es un problema tabular.
2. **Seleccionar un modelo adecuado:**
    - Para NLP: **BERT**, **GPT**, **LSTM**.
    - Para datos tabulares: **Redes neuronales densas**.
3. **Dividir datos:**
    - Separar en conjunto de entrenamiento, validación y prueba.
4. **Implementar Transfer Learning:**
    - Cargar un modelo preentrenado y añadir una capa de clasificación específica.
5. **Entrenar y evaluar:**
    - Usar una función de pérdida como *Cross-Entropy Loss*.
    - Medir métricas como *Accuracy*, *F1-Score*.
6. **Optimizar:**
    - Ajustar hiperparámetros y evitar *overfitting* con técnicas como *Dropout* o *Early Stopping*.

---

8. Tengo un problema de NER (Reconocimiento de Entidades Nombradas), tengo los datos, ¿cómo lo resolverías?

1. **Preprocesamiento:**
    - Convertir el texto a tokens y etiquetar cada token con su entidad (`PER`, `LOC`, `ORG`, `O`).
2. **Seleccionar un modelo preentrenado:**
    - Ejemplo: **BERT** o **RoBERTa**.
3. **Fine-Tuning:**
    - Ajustar el modelo a tus datos etiquetados.

Ejemplo en código con HuggingFace:
python
Copiar código

```
from transformers import BertForTokenClassification
```

```
# Cargar BERT para NER

model = BertForTokenClassification.from_pretrained('bert-base-uncased', num_labels=9)
```

4.
5. **Entrenamiento:**
    - Usar una función de pérdida como *Cross-Entropy Loss*.
    - Dividir datos en entrenamiento, validación y prueba.
6. **Evaluación:**
    - Medir métricas como *F1-Score*, *Precision*, y *Recall*.

---

8. ¿Qué se aprende durante el *pre-training*?

Durante el **pre-training**, el modelo aprende representaciones generales del lenguaje a partir de grandes cantidades de datos no etiquetados. Se capturan:

1. **Patrones semánticos y sintácticos:**
    - Ejemplo: Relaciones entre palabras como “perro” y “ladrar”.
2. **Contexto:**
    - Dependencias entre palabras y su significado según el contexto.
3. **Estructura gramatical:**
    - El orden y la relación gramatical entre palabras en una oración.
4. **Relaciones a largo plazo:**
    - Ejemplo: Conectar sujetos con predicados que están separados por muchas palabras.

---

9. ¿Qué es un MLM (*Masked Language Model*)?

Un **MLM (Masked Language Model)** es una técnica de preentrenamiento donde ciertas palabras en una secuencia se enmascaran y el modelo debe predecirlas.

Ejemplo:

- Frase original: `El perro corre rápido`.
- Frase enmascarada: `El [MASK] corre rápido`.
- Objetivo del modelo: Predecir `perro`.

Ejemplo de modelo:

- **BERT** usa MLM para aprender representaciones bidireccionales del lenguaje.

---

10. ¿Cómo se preentrena un modelo unidireccional?

Un modelo **unidireccional** se preentrena prediciendo palabras en una secuencia usando únicamente el contexto anterior (hacia la izquierda).

Ejemplo:
Para la frase `El perro corre`, el modelo predice `corre` utilizando solo `El` y `perro`.

Modelo representativo:

- **GPT** (Generative Pre-trained Transformer).

---

11. Explica el mecanismo de atención, cómo funciona, intuición, etc.

El **mecanismo de atención** permite que un modelo enfoque su atención en diferentes partes de una secuencia al procesar un token específico. Es como si el modelo decidiera “a qué palabras prestar más atención”.

Intuición:
Cuando leemos una oración, al interpretar una palabra, a veces nos enfocamos en otras palabras que le dan contexto.

Funcionamiento matemático:

1. **Inputs:** Vectores de consulta (**Q**), clave (**K**) y valor (**V**).

2. **Cálculo de pesos:** Se mide la similitud entre **Q** y **K**.
3. **Softmax:** Se normalizan los pesos para obtener probabilidades.
4. **Ponderación:** Se combina **V** usando los pesos calculados.

Attention(Q,K,V)=softmax(QKTdk)V\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)VAttention(Q,K,V)=softmax(dkQKT)V

---

12. ¿Qué es *Self-Attention*?

Self-Attention (Atención Autosupervisada) es una variante del mecanismo de atención donde cada token en una secuencia se compara con todos los demás (incluyéndose a sí mismo) para capturar relaciones contextuales.

Ejemplo:
En la oración `El gato persigue al ratón`, al procesar `persigue`, *Self-Attention* puede enfocarse en `gato` y `ratón` para entender mejor el contexto.

---

13. Explica la arquitectura *Transformers*. ¿Cómo se compara con una RNN? ¿Pros y cons?

La arquitectura **Transformer** está compuesta por bloques de *encoders* y *decoders* que utilizan el mecanismo de atención.

Componentes de un *Transformer*:

1. **Encoder:**
    - Procesa la secuencia de entrada.
    - Incluye *Self-Attention* y redes *Feed-Forward*.
2. **Decoder:**
    - Genera la salida.
    - Utiliza *Self-Attention* y *Cross-Attention*.

Comparación con RNN:

Característica	Transformer	RNN
Paralelización	Alta (procesa tokens simultáneamente)	Baja (procesa secuencia paso a paso)
Dependencias largas	Maneja dependencias largas fácilmente	Dificultades con dependencias largas
Atención	Usa *Self-Attention* global	Atención secuencial
Eficiencia	Mejor en GPUs	Más lento para secuencias largas
Complejidad	Mayor complejidad computacional	Menor complejidad

Pros de *Transformers*:

- Procesan secuencias de manera paralela.
- Capturan dependencias a larga distancia.

Contras de *Transformers*:

- Mayor consumo de memoria y cómputo.
- Necesitan más datos para entrenar.

---

14. ¿Qué es el *encoder*?

El **encoder** es una parte de la arquitectura *Transformer* que procesa la secuencia de entrada para generar representaciones contextuales.

Estructura del *encoder*:

1. **Self-Attention:** Captura relaciones entre los tokens de entrada.
2. **Feed-Forward Network (FFN):** Capas densas aplicadas individualmente a cada token.
3. **Normalización y *Residual Connections*:** Facilitan el entrenamiento.

---

15. ¿Cuál es la diferencia entre una red bidireccional y una unidireccional?

- **Unidireccional:**
  - Procesa la secuencia solo en una dirección (por ejemplo, de izquierda a derecha).
  - Ejemplo: **GPT**.
- **Bidireccional:**
  - Procesa la secuencia en ambas direcciones simultáneamente.
  - Ejemplo: **BERT**.

Diferencia clave:

- **Bidireccional** permite considerar el contexto tanto anterior como posterior a una palabra.
- **Unidireccional** solo considera el contexto previo, lo cual es útil para tareas generativas.

---

14. ¿Cuál es la diferencia conceptual entre *word2vec* y *GloVe*?

- **Word2Vec:**
  - Es un modelo basado en redes neuronales que aprende representaciones de palabras a partir del contexto de las mismas en una ventana fija.
  - Se entrena con métodos como **Skip-Gram** o **Continuous Bag-of-Words (CBOW)**.
  - Es **predictivo**, es decir, predice palabras basadas en su contexto.
- **GloVe (Global Vectors):**
  - Utiliza estadísticas globales de co-ocurrencia de palabras para aprender las representaciones.

    - Es un modelo **basado en matrices de co-ocurrencia** que se factoriza para obtener los *embeddings*.
    - Es **conteo**-basado, no predictivo.

Resumen:

- *Word2Vec* se enfoca en el contexto local de palabras (ventanas pequeñas).
- *GloVe* se basa en co-ocurrencias globales de palabras en todo el corpus.

---

15. ¿Qué es un modelo de lenguaje?

Un **modelo de lenguaje** es un sistema que asigna una probabilidad a una secuencia de palabras y predice palabras futuras en función del contexto previo.

Ejemplo:
Dado el texto: `El gato está en el...`, el modelo puede predecir `techo`, `jardín`, etc.

---

16. ¿Qué es un modelo de lenguaje unidireccional?

Un **modelo de lenguaje unidireccional** predice una palabra basándose únicamente en el contexto previo (hacia la izquierda).

Ejemplo:
Modelos como **GPT** son unidireccionales: para predecir una palabra en `El perro corre`, solo usan `El` y `perro`.

---

17. ¿Qué es un modelo de lenguaje generativo?

Un **modelo de lenguaje generativo** es capaz de generar texto nuevo a partir de una entrada o una semilla inicial. Se entrena para predecir y generar secuencias plausibles.

Ejemplos:

- **GPT** (Generative Pre-trained Transformer).
- **LLaMA**, **ChatGPT**.

---

18. ¿Qué es un modelo de lenguaje *n-gram*?

Un **modelo *n-gram*** predice una palabra basándose en las *n-1* palabras anteriores.

Ejemplo:

- Un modelo **bigram** predice una palabra basada en la anterior: `El perro`.
- Un modelo **trigram** predice una palabra basada en las dos anteriores: `El perro corre`.

---

19. ¿Cuáles son las principales desventajas de un modelo *n-gram*?

1. **Escalabilidad:**
    - El número de *n-grams* posibles crece exponencialmente con *n*.
2. **Datos limitados:**
    - Necesitan grandes corpus para cubrir todas las combinaciones posibles.
3. **Memoria:**
    - Requieren almacenar grandes tablas de frecuencia.
4. **Contexto limitado:**
    - No capturan dependencias a largo plazo debido a su ventana fija.

---

20. ¿Cuál es la importancia del vocabulario?

- **Cobertura:** Un vocabulario amplio asegura que el modelo reconozca más palabras del corpus.
- **Eficiencia:** Vocabularios más pequeños permiten modelos más rápidos y eficientes.

- **Generalización:** Un vocabulario adecuado ayuda a evitar problemas como palabras fuera del vocabulario (*OOV*).

---

21. ¿Qué es la tokenización?

La **tokenización** es el proceso de dividir un texto en unidades más pequeñas llamadas **tokens**. Estos pueden ser palabras, subpalabras o caracteres.

Ejemplos:

- **Tokenización por palabras:** `El perro corre` → `[El, perro, corre]`
- **Tokenización por subpalabras:** `Corriendo` → `[Cor, riendo]`

---

22. ¿Cómo se pueden evaluar los modelos de lenguaje?

1. **Perplexity (Perplejidad):**
   - Mide qué tan bien el modelo predice una secuencia. Valores más bajos son mejores.
2. **BLEU (Bilingual Evaluation Understudy):**
   - Se usa para tareas de traducción automática.
3. **ROUGE (Recall-Oriented Understudy for Gisting Evaluation):**
   - Evalúa la calidad del resumen de texto.
4. **Exact Match / F1-Score:**
   - Para tareas como clasificación o preguntas y respuestas.

---

23. ¿Qué relación hay entre *word embeddings* y redes neuronales? ¿Por qué funcionan bien juntos?

- **Relación:**
  - Los *word embeddings* son representaciones vectoriales aprendidas a partir de redes neuronales.
  - Se utilizan como entrada para redes neuronales en tareas de NLP.
- **Compatibilidad:**

    - Las redes neuronales pueden aprender patrones complejos en los vectores de *word embeddings*, capturando relaciones semánticas entre palabras.

---

24. ¿Qué es ELMo y cuál es su arquitectura?

- **ELMo (Embeddings from Language Models)** es un modelo que produce *embeddings* contextuales para palabras.
- Utiliza una **arquitectura de dos capas bidireccionales LSTM** entrenadas con un modelo de lenguaje.

Características:

- Los *embeddings* cambian según el contexto.
- Captura información sintáctica y semántica.

---

25. ¿Cómo diseñarías un problema de clasificación con ELMo?

1. **Dataset:**
    - Ejemplo: Clasificación de sentimiento en reseñas (`Positivo`, `Negativo`).
2. **Tokenización:**
    - Tokenizar el texto en palabras.
3. **Obtención de *embeddings* con ELMo:**
    - Utilizar el modelo ELMo para obtener *embeddings* contextuales para cada palabra.
4. **Promediado:**
    - Promediar los *embeddings* de las palabras para obtener una representación de la oración.
5. **Clasificador:**
    - Pasar los *embeddings* promediados a una red neuronal densa para clasificar.
6. **Entrenamiento:**
    - Entrenar el clasificador con una función de pérdida como *Cross-Entropy Loss*.

---

25. ¿Qué es el paradigma de *pre-training* y *fine-tuning*?

- **Pre-training:**
  Es el entrenamiento inicial de un modelo en una tarea general usando grandes cantidades de datos no etiquetados. Ejemplo: BERT se preentrena con *Masked Language Modeling (MLM)*.
- **Fine-tuning:**
  Es el ajuste del modelo preentrenado en una tarea específica con datos etiquetados. Ejemplo: Ajustar BERT para clasificación de texto.

Ejemplo de flujo:

1. **Pre-training:** Entrenar BERT en grandes corpus de texto para aprender representaciones generales del lenguaje.
2. **Fine-tuning:** Ajustar BERT en un dataset específico para clasificación de sentimiento.

---

26. ¿Qué es el mecanismo de atención? ¿Cuáles son sus ventajas?

El **mecanismo de atención** permite a un modelo enfocarse en diferentes partes de una secuencia al procesar datos. En lugar de tratar todos los tokens por igual, la atención asigna pesos a cada token según su relevancia para la tarea.

Ventajas:

1. **Paralelización:** Procesa tokens simultáneamente, a diferencia de RNNs que procesan secuencias de manera secuencial.
2. **Relaciones a largo plazo:** Captura dependencias a larga distancia sin degradación de información.
3. **Flexibilidad:** Puede aplicarse a diversas tareas como traducción, clasificación y generación de texto.

---

27. ¿Cómo relacionas el mecanismo de atención con la arquitectura *Transformers*?

El mecanismo de atención es la **base fundamental** de los *Transformers*. La arquitectura utiliza **Self-Attention** (Atención Autosupervisada) para permitir que cada token en una secuencia atienda a todos los demás tokens simultáneamente.

Ejemplo en *Transformers*:

- En el *encoder*, cada token de entrada utiliza *Self-Attention* para obtener una representación contextualizada basada en toda la secuencia.
- En el *decoder*, se aplica tanto *Self-Attention* como *Cross-Attention* (entre las representaciones del *encoder* y el *decoder*).

---

28. ¿Qué ventajas tienen los *Transformers* frente a las redes neuronales recurrentes (RNN)?

1. **Paralelización:**
    - *Transformers* pueden procesar todos los tokens simultáneamente, mientras que las RNN procesan secuencias de forma secuencial.
2. **Eficiencia computacional:**
    - Gracias a la paralelización, los *Transformers* son más rápidos y eficientes en hardware como GPUs.
3. **Mejor manejo de dependencias a largo plazo:**
    - Las RNN sufren con dependencias largas debido al problema del *vanishing gradient*. Los *Transformers* pueden manejar contextos largos sin este problema.
4. **Escalabilidad:**
    - Los *Transformers* pueden escalar fácilmente a modelos con billones de parámetros (Ej., GPT-3, BERT).

---

29. ¿Podés hacer un *overview* de la arquitectura *Transformers*?

La arquitectura *Transformer* se compone de dos bloques principales:

1. **Encoder:**
    - Compuesto por varias capas que incluyen:
        - *Self-Attention* (Atención Autosupervisada).

        - *Feed-Forward Neural Network* (FFN).
        - *Layer Normalization* y *Residual Connections*.
2. **Decoder:**
    - Similar al *encoder* pero con una capa adicional de *Cross-Attention* para combinar información del *encoder*.

Resumen de componentes:

- **Multi-Head Attention:** Permite que el modelo atienda a diferentes aspectos de la entrada simultáneamente.
- **Positional Embeddings:** Añaden información de posición a los tokens.
- **Feed-Forward Networks:** Capas densas aplicadas a cada token independientemente.

---

30. ¿Cómo soluciona el *Transformer* la falta de sentido posicional (opuesto a las RNN)?

Los *Transformers* no procesan secuencias de manera secuencial, por lo que no capturan automáticamente el orden de los tokens. Para solucionar esto, utilizan **Positional Embeddings** (embeddings posicionales) que se suman a los embeddings de los tokens para indicar su posición en la secuencia.

---

31. ¿Qué son los *Positional Embeddings*?

Son vectores que representan la posición de cada token en una secuencia. Estos se suman a los embeddings de los tokens para que el modelo tenga información sobre el orden de los tokens.

Tipos de *Positional Embeddings*:

1. **Fijos (senos y cosenos):** Utilizados en el *Transformer* original.
2. **Aprendibles:** Utilizados en modelos como BERT y GPT.

---

32. ¿Qué es *Self-Attention*?

Self-Attention permite a un token en una secuencia atender a todos los demás tokens y a sí mismo para capturar relaciones contextuales.

Fórmula básica:

Attention(Q,K,V)=softmax(QKTdk)V\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)VAttention(Q,K,V)=softmax(dkQKT)V

- **Q (Query):** Representación del token que consulta.
- **K (Key):** Representación de los tokens consultados.
- **V (Value):** Representación que se devuelve como resultado.

---

33. ¿Qué es el *encoder*?

El **encoder** es una parte de la arquitectura *Transformer* que procesa la secuencia de entrada y genera representaciones contextuales. Está compuesto por múltiples capas con:

1. **Self-Attention:** Captura relaciones entre tokens.
2. **Feed-Forward Network (FFN):** Capas densas aplicadas a cada token.
3. **Normalización y Residual Connections:** Facilitan el entrenamiento profundo.

---

34. Describí el *input* de BERT.

1. **Tokens de entrada:**
    - Se tokeniza el texto y se agregan tokens especiales `[CLS]` (inicio) y `[SEP]` (separador).
2. **Segment Embeddings:**
    - Indican si el token pertenece a la primera o segunda oración (usado en *Next Sentence Prediction*).
3. **Positional Embeddings:**
    - Representan la posición de cada token.
4. **Embeddings finales:**

- Suma de **Token Embeddings**, **Segment Embeddings**, y **Positional Embeddings**.

---

35. Describí el proceso de *MLM* (Masked Language Modeling).

1. **Enmascaramiento:**
   - Se selecciona aleatoriamente un 15% de los tokens de la secuencia y se reemplazan por:
     - `[MASK]` (80% de los casos).
     - Otro token aleatorio (10%).
     - El mismo token original (10%).
2. **Predicción:**
   - El modelo predice el token original en las posiciones enmascaradas.
3. **Entrenamiento:**
   - Se usa una función de pérdida para medir qué tan bien predice el modelo los tokens enmascarados.

---

36. Diseña una tarea de clasificación con BERT.

Ejemplo: Clasificación de sentimiento en reseñas de películas.

1. **Dataset:**
   - Utilizar un dataset como `IMDb` con reseñas etiquetadas como `Positivo` o `Negativo`.
2. **Preparación de datos:**
   - Tokenizar las reseñas con el *tokenizer* de BERT.
   - Añadir tokens especiales `[CLS]` y `[SEP]`.
   - Asegurarse de que las secuencias tengan la misma longitud (`padding`).
3. **Modelo:**
   - Usar `BERTForSequenceClassification` de Hugging Face (`transformers`).
   - Esta clase agrega una capa de clasificación encima de BERT.
4. **Entrenamiento:**

    - Entrenar el modelo con un optimizador como `AdamW`.
    - Utilizar una función de pérdida como `CrossEntropyLoss`.
5. **Evaluación:**
    - Evaluar el modelo con métricas como `accuracy` o `F1-score`.

---

37. ¿Cómo harías para anonimizar entidades de fallos judiciales?

1. **Reconocimiento de Entidades (NER):**
    - Utilizar un modelo de NER (por ejemplo, `spaCy` o `BERT-NER`) para identificar entidades como nombres, fechas, lugares, y números de identificación.
2. **Sustitución de entidades:**
    - Reemplazar las entidades con etiquetas genéricas:
        - Nombres → `[NOMBRE]`
        - Fechas → `[FECHA]`
        - Lugares → `[LUGAR]`
        - Identificadores → `[ID]`
3. **Revisión manual:**
    - Realizar una revisión para asegurar que no queden entidades sin anonimizar.
4. **Automatización con scripts:**
    - Desarrollar un script en Python que combine NER y sustitución automática.

---

38. ¿Qué diferencia hay entre un modelo bidireccional y uno unidireccional? ¿Cómo se manifiesta la diferencia en términos de *attention*?

- **Modelo Bidireccional (Ej., BERT):**
    - Considera el contexto a la izquierda y a la derecha del token actual simultáneamente.
    - **Attention:** Cada token puede atender a cualquier otro token de la secuencia.

- **Modelo Unidireccional (Ej., GPT):**
    - Considera solo el contexto hacia la izquierda del token actual.
    - **Attention:** La atención está enmascarada para evitar que un token atienda a futuros tokens.

Ejemplo:
Para predecir la palabra `comer` en:
`Voy a [MASK] manzanas.`

- **BERT** puede usar `Voy a` y `manzanas` para predecir `[MASK]`.
- **GPT** solo puede usar `Voy a` porque no ve `manzanas`.

---

39. ¿Qué es una arquitectura *encoder-decoder*?

Es una arquitectura que se utiliza principalmente en tareas de transformación de secuencias como traducción automática.

- **Encoder:** Codifica la secuencia de entrada en una representación intermedia.
- **Decoder:** Toma esta representación y genera la secuencia de salida.

Ejemplo:
En traducción de español a inglés:

- **Entrada:** `Hola, ¿cómo estás?`
- **Encoder:** Genera una representación intermedia.
- **Decoder:** Produce `Hello, how are you?`

---

40. ¿Qué es el *pre-training*? ¿Qué relación tiene con el *transfer learning*?

- **Pre-training:**
  Entrenamiento inicial de un modelo en una gran cantidad de datos no etiquetados con tareas generales (Ej., *Masked Language Modeling* para BERT).

- **Transfer Learning:**
  Aprovechar el modelo preentrenado y ajustarlo (*fine-tuning*) a una tarea específica (Ej., clasificación de texto).

Relación:
El *pre-training* crea una base de conocimiento general, y el *transfer learning* permite especializar ese conocimiento para tareas concretas.

---

41. ¿En qué capas se centra el *pre-training* y en cuáles el *fine-tuning*?

- **Pre-training:**
  Afecta todas las capas del modelo, ya que se entrena desde cero para tareas generales.
- **Fine-tuning:**
  Ajusta las últimas capas del modelo para adaptarlo a una tarea específica, aunque también se pueden ajustar todas las capas si se requiere.

---

42. ¿Qué tipos de preentrenamiento conoces?

1. **Masked Language Modeling (MLM):**
   Ej., BERT: Predecir palabras ocultas en una secuencia.
2. **Autoregressive Language Modeling:**
   Ej., GPT: Predecir el siguiente token dado el contexto anterior.
3. **Next Sentence Prediction (NSP):**
   Ej., BERT: Determinar si dos frases son consecutivas.
4. **Denoising Autoencoders:**
   Ej., BART: Reconstruir una secuencia corrupta.
5. **Sequence-to-Sequence (Seq2Seq):**
   Ej., T5: Predecir una transformación de texto.

---

43. ¿Qué información incorpora el preentrenamiento?

- **Sintaxis y gramática del lenguaje.**
- **Semántica y significado de palabras.**

- **Relaciones entre palabras y frases.**
- **Conocimiento del mundo (en modelos grandes).**

---

45. Explica conceptualmente el paradigma *pre-training* (PT) y *fine-tuning* (FT).

- **Pre-training (PT):**
  - Entrenar un modelo en una tarea general con grandes cantidades de datos.
  - Ej., BERT se preentrena con *Masked Language Modeling*.
- **Fine-tuning (FT):**
  - Ajustar el modelo preentrenado en una tarea específica usando datos etiquetados.
  - Ej., Ajustar BERT para clasificación de sentimientos.

---

46. ¿Qué son los *adapters*?

Son módulos adicionales que se insertan en las capas de un modelo preentrenado para permitir el *fine-tuning* con un número reducido de parámetros.

Características:

- **Eficiencia:** Solo se ajustan los parámetros de los *adapters*.
- **Modularidad:** Se pueden tener *adapters* para diferentes tareas sin modificar el modelo base.
- **Ejemplo:** LoRA es una variante de *adapters* que usa matrices de bajo rango.

---

47. ¿Qué es *LoRA*?

LoRA (Low-Rank Adaptation) es una técnica de *fine-tuning* eficiente para modelos de lenguaje grandes. En lugar de ajustar todos los parámetros del modelo, LoRA inserta matrices de bajo rango en las capas de atención y solo ajusta estos componentes adicionales durante el entrenamiento.

Ventajas de LoRA:

- **Menor uso de memoria:** Se ajusta un número reducido de parámetros.
- **Eficiencia:** Permite entrenar con menos recursos computacionales.
- **Flexibilidad:** Fácil de aplicar a modelos preentrenados sin modificar su estructura original.

---

48. ¿Qué es un *encoder-decoder*?

Un **encoder-decoder** es una arquitectura utilizada en tareas de procesamiento de lenguaje natural como traducción automática. Se compone de dos partes:

1. **Encoder:**
   Procesa la secuencia de entrada y genera representaciones (embeddings) contextuales.
2. **Decoder:**
   Toma las representaciones generadas por el *encoder* y produce una secuencia de salida.

Ejemplos:

- **Transformers como BART y T5** utilizan esta arquitectura.
- En traducción, el *encoder* procesa una frase en español y el *decoder* genera la traducción en inglés.

---

49. ¿Cómo se preentrena cada arquitectura?

1. **Encoder-Only (Ej., BERT):**
   - **Tarea de preentrenamiento:** *Masked Language Modeling (MLM)*.
   - Objetivo: Predecir palabras ocultas en una oración.
2. **Decoder-Only (Ej., GPT):**
   - **Tarea de preentrenamiento:** *Autoregressive Language Modeling*.
   - Objetivo: Predecir el siguiente token en una secuencia dada.
3. **Encoder-Decoder (Ej., T5):**
   - **Tarea de preentrenamiento:** *Seq2Seq* con transformaciones de texto como reescritura, resumen o traducción.

- Objetivo: Aprender a transformar una entrada en una salida específica.

---

50. ¿Qué son las capacidades emergentes?

Las **capacidades emergentes** son habilidades que no están presentes en modelos pequeños pero aparecen cuando se entrena un modelo con una cantidad de parámetros o datos suficientemente grande. Estas habilidades no son explícitamente programadas, sino que emergen del aprendizaje automático del modelo.

Ejemplos:

- Comprensión de instrucciones complejas.
- Resolución de problemas matemáticos complejos.
- Traducción multilingüe sin entrenamiento específico para todos los idiomas.

---

51. ¿Cómo surgen las capacidades emergentes?

Surgen debido a:

1. **Escalamiento del modelo:**
   A medida que el modelo aumenta en tamaño (más parámetros), su capacidad para capturar patrones complejos también aumenta.
2. **Escalamiento de datos:**
   Entrenar con grandes cantidades de datos diversos permite al modelo aprender una amplia gama de patrones.
3. **Complejidad del entrenamiento:**
   El modelo descubre representaciones más abstractas y generalizables cuando se entrena con tareas variadas.

---

52. ¿Qué es *In-Context Learning*?

In-Context Learning (ICL) es la capacidad de un modelo de lenguaje para aprender patrones o instrucciones a partir del contexto proporcionado en el *prompt*, sin actualizar sus parámetros.

Ejemplo:
Dado el *prompt*:

rust

Copiar código

```
Traduce lo siguiente del inglés al español:

"Hello" -> "Hola"

"Goodbye" -> "Adiós"

"Thank you" ->
```

El modelo deduce que debe traducir "Thank you" a "Gracias".

---

53. ¿Qué es *Zero-Shot* y *Few-Shot*?

- **Zero-Shot:**
  El modelo realiza una tarea sin ejemplos previos en el *prompt*.
  Ejemplo:
  `Traduce "dog" al español.`

Few-Shot:
El modelo realiza una tarea con algunos ejemplos proporcionados en el *prompt*.
Ejemplo:
rust
Copiar código
`"dog" -> "perro"`

```
"cat" -> "gato"

"bird" ->
```

-

---

54. ¿Qué son las alucinaciones?

Las **alucinaciones** son respuestas generadas por un modelo que son plausibles pero incorrectas o irrelevantes respecto a la realidad o al contexto. Ocurren cuando el modelo inventa información.

Ejemplo:
Afirmar que "París es la capital de Japón" es una alucinación.

---

55. ¿Cómo se mitigan las alucinaciones?

1. **Incorporar datos de alta calidad:**
   Aumentar el uso de datos precisos y relevantes durante el entrenamiento.
2. **Fine-tuning específico:**
   Ajustar el modelo con datos específicos de la tarea para mejorar su precisión.
3. **RAG (Retrieval-Augmented Generation):**
   Incorporar sistemas de recuperación de información externa para respaldar las respuestas.
4. **Filtrado de respuestas:**
   Utilizar modelos adicionales (como *Reward Models*) para evaluar y filtrar las respuestas generadas.

---

56. ¿Qué es un *RAG*?

RAG (Retrieval-Augmented Generation) es una técnica que combina generación de lenguaje con recuperación de información externa. El modelo

genera respuestas basadas en documentos o datos recuperados de una base de datos o corpus externo.

Proceso:

1. Se recupera información relevante para una consulta.
2. El modelo de lenguaje usa esta información para generar una respuesta más precisa y fundamentada.

---

57. ¿Cómo se pasa de un modelo base a un asistente?

1. **Fine-tuning supervisado:**
   Ajustar el modelo base con datos de diálogos y tareas específicas.
2. **Instruction Tuning:**
   Entrenar el modelo para seguir instrucciones explícitas.
3. **RLHF (Reinforcement Learning from Human Feedback):**
   Mejorar el comportamiento del modelo utilizando retroalimentación humana para alinear sus respuestas.
4. **Incorporación de herramientas adicionales:**
   Integrar recuperación de información (RAG), APIs, o sistemas de verificación.

---

58. ¿Qué tipo de *prompts* funcionan en los modelos base: *Zero-Shot* o *Few-Shot*? ¿Por qué?

- **Few-Shot:**
  Los modelos base suelen funcionar mejor con *few-shot prompting* porque no están específicamente ajustados para seguir instrucciones. Proporcionar ejemplos ayuda al modelo a entender la tarea.

Razón:
El modelo base fue entrenado para predecir el siguiente token en textos generales, no para seguir instrucciones. Los ejemplos en el *prompt* le dan un contexto explícito de la tarea esperada.

---

59. ¿Qué es el *Instruction Tuning* y cómo se diferencia del *Pretraining (PT)*?

- **Instruction Tuning** es una técnica para ajustar modelos de lenguaje (LLMs) para que respondan de manera más efectiva a instrucciones específicas dadas por los usuarios. Se enfoca en afinar un modelo preentrenado utilizando ejemplos de tareas formuladas como instrucciones (por ejemplo: *"Resume el siguiente texto"*).
- **Pretraining (PT)**, en cambio, es el entrenamiento inicial de un modelo en grandes volúmenes de datos no estructurados (generalmente corpus de texto masivos). El objetivo del *pretraining* es que el modelo aprenda representaciones del lenguaje y patrones generales, como sintaxis, semántica y relaciones contextuales.

Diferencias clave:

- **PT** es aprendizaje no supervisado o auto-supervisado en grandes volúmenes de texto para generar una base de conocimiento general.
- **Instruction Tuning** es un proceso supervisado que ajusta el modelo para seguir instrucciones de manera más específica y útil.

---

60. ¿Cómo se pueden coleccionar los datos para *Instruction Tuning*?

Los datos para *Instruction Tuning* pueden recolectarse de varias maneras:

1. **Anotación manual:** Se crean pares de *instrucción-respuesta* con la ayuda de anotadores humanos.
2. **Extracción de datasets existentes:** Recolectar datos de datasets públicos que contienen tareas específicas, como:
    - Datasets de tareas de procesamiento del lenguaje natural (NLP) como SQuAD, SuperGLUE, o datasets de preguntas y respuestas.
3. **Interacción con usuarios reales:** Aprovechar las interacciones con un sistema en producción para recolectar instrucciones y respuestas relevantes.

---

61. ¿Se pueden coleccionar sintéticamente? ¿Cómo?

Sí, se pueden generar datos sintéticos para *Instruction Tuning* de las siguientes maneras:

1. **Generación con otros LLMs:** Utilizar modelos preexistentes como GPT-3 o GPT-4 para generar pares de *instrucción-respuesta*. Por ejemplo, se le puede pedir a un modelo grande que genere instrucciones variadas y sus correspondientes respuestas.
2. **Paráfrasis y reformulación:** Crear múltiples versiones de una misma instrucción mediante técnicas de reformulación automática.

3. **Auto-generación y auto-refinamiento:**
    - Generar una instrucción inicial y luego usar el modelo para producir respuestas.
    - Refinar la instrucción y respuesta mediante validación automática o feedback de otros modelos.
4. **Prompt engineering y programación automática:** Diseñar *prompts* específicos que produzcan una amplia gama de instrucciones y tareas en un formato deseado.

---

62. ¿Qué es el *RLHF*? ¿Para qué sirve?

- **Reinforcement Learning from Human Feedback (RLHF)** es una técnica para entrenar modelos de lenguaje utilizando retroalimentación humana para alinear mejor las respuestas del modelo con las expectativas humanas.
- **Objetivo:** Optimizar el comportamiento del modelo para que genere respuestas más útiles, precisas y seguras, evitando problemas como respuestas sesgadas o dañinas.

Proceso general:

1. Se recopilan ejemplos de respuestas generadas por el modelo.
2. Humanos califican o clasifican estas respuestas según su calidad.
3. Esta retroalimentación se utiliza para ajustar el modelo mediante técnicas de *aprendizaje por refuerzo*.

---

63. ¿Cómo se entrena un *Reward Model* en el contexto de *RLHF*? ¿Cuál es su función?

1. **Recopilación de datos:**
    - Se recopilan múltiples respuestas del modelo para una misma instrucción.
    - Humanos clasifican estas respuestas por calidad (ordenándolas del mejor al peor).
2. **Entrenamiento del Reward Model:**
    - Se entrena una red neuronal (el *Reward Model*) para predecir una puntuación o *reward* que refleje qué tan buena es una respuesta dada una instrucción.
    - Este modelo se entrena con pares de respuestas clasificadas para que aprenda a asignar puntuaciones coherentes con el juicio humano.

Función:
El *Reward Model* proporciona la señal de recompensa durante el proceso de *aprendizaje por refuerzo*. Ayuda a ajustar el modelo de lenguaje para que sus respuestas maximicen esta recompensa, alineándose mejor con los deseos humanos.

---

64. Explica *RLHF*. ¿Cuál es el rol de *PPO*? ¿Cómo se implementa?

Proceso de RLHF:

1. **Preentrenamiento del modelo de lenguaje:** Se entrena el modelo en grandes cantidades de datos no supervisados.
2. **Instruction Tuning:** Se ajusta el modelo para que siga instrucciones específicas.
3. **Entrenamiento del Reward Model:** Se entrena un modelo de recompensa para evaluar la calidad de las respuestas generadas.
4. **Optimización por refuerzo (PPO):** Se ajusta el modelo utilizando *Proximal Policy Optimization (PPO)* para maximizar las recompensas dadas por el *Reward Model*.

Rol de PPO:

- **PPO** es un algoritmo de *aprendizaje por refuerzo* que ajusta el modelo de manera estable y eficiente.
- **Ventajas:**
    - Mantiene las actualizaciones del modelo dentro de límites seguros para evitar grandes desviaciones del comportamiento previo.
    - Es eficiente computacionalmente y adecuado para entrenar grandes modelos.

Implementación:
Se implementa con bibliotecas como `transformers` y `RL frameworks` como `Stable Baselines` o `RLlib`.

---

65. ¿Te acuerdas cómo se lleva LLaMA2 desde el modelo base hasta el chat? ¿Algún componente novedoso en relación al ChatGPT?

Proceso de llevar LLaMA2 a un modelo tipo chat:

1. **Preentrenamiento:** LLaMA2 se entrena inicialmente con grandes cantidades de texto para aprender representaciones generales del lenguaje.
2. **Instruction Tuning:** Se ajusta LLaMA2 con datos de instrucciones y respuestas para convertirlo en un asistente conversacional.
3. **RLHF:** Se aplica *Reinforcement Learning from Human Feedback* para afinar aún más el modelo y asegurar que sus respuestas estén alineadas con las expectativas humanas.

Componentes novedosos en comparación con ChatGPT:

- **LLaMA2 está optimizado para ser eficiente y accesible:**
  Se enfoca en modelos más ligeros (7B a 70B parámetros), comparado con los modelos más grandes de OpenAI.

- **Uso abierto:** LLaMA2 es accesible para investigación y uso comercial, mientras que ChatGPT (GPT-4) es de código cerrado.
- **Entrenamiento con datos más transparentes:** Meta ha proporcionado más detalles sobre el proceso de entrenamiento y ajuste en comparación con OpenAI.

---

66. ¿Qué es un *Reward Model*?

Un **Reward Model (RM)** es una red neuronal entrenada para asignar una puntuación (*reward*) a las respuestas generadas por un modelo de lenguaje. Este puntaje refleja qué tan bien se ajusta una respuesta a las preferencias humanas o a ciertos criterios deseados (por ejemplo, utilidad, seguridad, claridad).

Aplicaciones:

- Se usa principalmente en técnicas como **RLHF** (Reinforcement Learning from Human Feedback).
- Ayuda a ajustar el modelo para que maximice el *reward*, mejorando así su alineación con las expectativas humanas.

---

67. ¿Qué rol juega el *Rejection Sampling* en LLaMA?

Rejection Sampling se utiliza en LLaMA y otros modelos para **mejorar la calidad de las respuestas generadas** al filtrar muestras no deseadas. El proceso es:

1. **Generar múltiples respuestas (muestras)** para una misma instrucción usando el modelo.
2. **Evaluarlas** con un *Reward Model*.
3. **Seleccionar la mejor muestra** según la puntuación del *Reward Model*.

Ventaja:
Mejora el rendimiento sin necesidad de modificar directamente los parámetros del modelo. Se seleccionan respuestas de mayor calidad después de la generación.

---

68. ¿Qué es el *Chain of Thought*?

El **Chain of Thought (CoT)** es una técnica de razonamiento en modelos de lenguaje donde el modelo produce una secuencia de pasos intermedios al resolver una tarea compleja.

Ejemplo:
En lugar de responder directamente a una pregunta matemática compleja, el modelo desglosa el problema en pasos más pequeños y explica su razonamiento antes de llegar a la solución.

Beneficios:

- Mejora el rendimiento en tareas que requieren razonamiento lógico o multi-paso.
- Hace que el proceso del modelo sea más interpretable.

---

69. ¿Cómo se entrena un modelo para seguir instrucciones?

Proceso de entrenamiento para *Instruction Following*:

1. **Dataset de Instrucciones:** Crear o recopilar pares de *instrucción-respuesta*.
2. **Fine-tuning supervisado:** Ajustar un modelo preentrenado usando estos datos para que aprenda a generar respuestas siguiendo instrucciones.
3. **RLHF (opcional):** Mejorar aún más el comportamiento del modelo utilizando aprendizaje por refuerzo con retroalimentación humana.
4. **Evaluación continua:** Medir el desempeño del modelo en tareas específicas y ajustar según sea necesario.

---

70. Explicá la destilación de modelos.

La **destilación de modelos** es una técnica para transferir el conocimiento de un **modelo grande (teacher)** a un **modelo más pequeño (student)**. El objetivo es mantener el rendimiento del modelo original pero con una arquitectura más eficiente.

Proceso general:

1. El *teacher* genera predicciones o representaciones.
2. El *student* se entrena para imitar estas predicciones.
3. El *student* se optimiza para ser más rápido y ligero.

---

71. Explicá alguna técnica para destilar modelos.

Una técnica común es la **Destilación Clásica de Knowledge Distillation (KD)** de Hinton et al.:

1. **Generar *soft labels*:** El *teacher* produce distribuciones de probabilidad (*soft targets*) en lugar de respuestas binarias.

2. **Entrenar al *student* con una función de pérdida:** L=(1−α)LCE+αLKD\mathcal{L} = (1 - \alpha) \mathcal{L}_{CE} + \alpha \mathcal{L}_{KD}L=(1−α)LCE+αLKD Donde:
    - **LCE\mathcal{L}_{CE}LCE** es la pérdida de entropía cruzada con las etiquetas reales.
    - **LKD\mathcal{L}_{KD}LKD** es la pérdida entre las distribuciones del *teacher* y del *student*.
    - **α\alphaα** es un coeficiente para equilibrar ambas pérdidas.

---

72. ¿Qué pros y contras hay entre destilar un modelo por datos o hacerlo con un *teacher online*?

Método	Pros	Contras
Destilación por datos	- Simple de implementar. - Se pueden reutilizar predicciones del *teacher*.	- Limitado por los datos previamente generados. - Puede no generalizar bien.
Teacher online	- El *student* se entrena dinámicamente con las predicciones actualizadas del *teacher*.	- Requiere más recursos computacionales. - Más complejo de implementar.

---

73. ¿Qué es NAS y qué rol juega en la destilación de modelos?

NAS (Neural Architecture Search) es un proceso automático para encontrar arquitecturas óptimas de redes neuronales. Se utiliza para:

- **Optimizar modelos pequeños (students)** durante la destilación.
- Encontrar estructuras eficientes que mantengan el rendimiento del *teacher* pero con menor costo computacional.

Ejemplo de uso:
En destilación, NAS puede identificar la mejor arquitectura para el *student* que logre un equilibrio óptimo entre precisión y eficiencia.

---

74. Suponiendo que tenés una gran cantidad de datos sin anotar y una pequeña proporción anotada, ¿cómo generarías un modelo destilado?

1. **Preentrenamiento del modelo *student*:**
   Entrenar el modelo *student* con los datos sin anotar usando técnicas de aprendizaje auto-supervisado.
2. **Entrenamiento supervisado inicial:**
   Ajustar el *student* con la pequeña proporción de datos anotados para que aprenda las tareas específicas.
3. **Pseudo-etiquetado:**
   - Utilizar el *teacher* para generar etiquetas en los datos no anotados.
   - Entrenar el *student* con estos datos pseudo-etiquetados.
4. **Refinamiento con destilación:**
   Aplicar destilación con el *teacher* para ajustar el *student*, combinando las predicciones del *teacher* y las etiquetas anotadas.

---

75. ¿Por qué destilar modelos?

Razones para destilar modelos:

1. **Eficiencia computacional:**
   - Los modelos destilados son más pequeños y rápidos, lo que permite su despliegue en dispositivos con recursos limitados (móviles, IoT).
2. **Reducción de costos:**
   - Inferencia más rápida y menos costosa en términos de energía y tiempo.
3. **Escalabilidad:**
   - Facilita el despliegue masivo de modelos en producción.
4. **Mantenimiento de rendimiento:**
   - A pesar de su tamaño reducido, los modelos destilados pueden mantener una precisión cercana a la del *teacher*.

---

76. ¿Qué son los *soft labels* y *hard labels*? ¿Qué ventaja(s) da usar *soft labels*?

- **Hard Labels:**
  Son etiquetas binarias o categóricas tradicionales (por ejemplo, 0 o 1) utilizadas en el aprendizaje supervisado. Representan una única clase correcta para una entrada.
  Ejemplo:
  Si se clasifica una imagen como un "perro", la etiqueta sería `1` para "perro" y `0` para todas las demás clases.
- **Soft Labels:**
  Son distribuciones de probabilidad generadas por un modelo *teacher*. En lugar de asignar una única clase correcta, se asigna una probabilidad a cada clase posible.

Ejemplo:
Para una imagen de un perro que se parece un poco a un zorro, el *soft label* podría ser:
- Perro: 0.85
- Zorro: 0.10
- Gato: 0.05

Ventajas de usar *soft labels*:

1. **Información adicional:**
   Capturan relaciones entre clases, lo que ayuda al modelo *student* a generalizar mejor.
2. **Reducción de overfitting:**
   Evitan que el modelo se ajuste demasiado a respuestas categóricas rígidas.
3. **Regularización:**
   Actúan como una forma de regularización al permitir que el modelo *student* aprenda con incertidumbre en las predicciones.

---

77. ¿Cuál es la forma conocida más efectiva para destilar *transformers*?

Una de las técnicas más efectivas es **DistilBERT** de Hugging Face, que aplica una combinación de:

1. **Knowledge Distillation:**
   Se entrena un *student* para imitar el comportamiento del *teacher* utilizando *soft labels* generadas por el *teacher*.
2. **Reducción del número de capas:**
   DistilBERT reduce el número de capas del modelo a la mitad (por ejemplo, de 12 capas a 6) para mejorar la eficiencia.
3. **Triple pérdida de entrenamiento:**
   Combina tres tipos de pérdidas:
   - **Pérdida de distilación:** Para que el *student* imite las predicciones del *teacher*.
   - **Pérdida de entropía cruzada:** Con respecto a las etiquetas reales.
   - **Pérdida de *hidden states***: Para que el *student* aprenda representaciones similares a las del *teacher* en cada capa intermedia.

---

78. ¿Podés describir *AutoDistil*?

AutoDistil es un método automatizado para destilar modelos utilizando técnicas de **aprendizaje automático** y **búsqueda automatizada**. Su objetivo es encontrar de manera eficiente una arquitectura y configuración óptimas para el modelo *student*.

Principales características:

1. **NAS (Neural Architecture Search):**
   Utiliza búsqueda automatizada para encontrar la mejor arquitectura para el *student*.
2. **Generación automatizada de datos:**
   Puede generar datos sintéticos para entrenar el modelo *student*.
3. **Entrenamiento iterativo:**
   Entrena múltiples versiones del *student* con retroalimentación constante para seleccionar el mejor modelo destilado.

Ventaja:
Reduce el esfuerzo manual en el proceso de destilación y optimización.

---

79. ¿Cómo se están destilando los grandes modelos generativos?

La destilación de grandes modelos generativos implica varios enfoques avanzados para reducir su tamaño sin comprometer mucho su rendimiento:

1. **Knowledge Distillation Clásica:**
   Se entrena un modelo *student* más pequeño para imitar las predicciones del *teacher*, utilizando *soft labels* o representaciones intermedias.
2. **Destilación Secuencial (Sequence-Level Distillation):**
   El *teacher* genera secuencias completas (por ejemplo, textos completos), y el *student* se entrena para replicar estas secuencias.
3. **Destilación Progresiva:**
   Se destila el modelo en múltiples etapas, reduciendo gradualmente su tamaño.
4. **Rejection Sampling:**
   Se generan múltiples respuestas con el *teacher*, se filtran las mejores, y el *student* se entrena solo con estas respuestas de alta calidad.
5. **RLHF + Destilación:**
   Combinar *Reinforcement Learning from Human Feedback (RLHF)* con destilación para producir un modelo alineado y eficiente.

---

80. ¿Qué técnicas conocés para destilar grandes modelos generativos?

1. **Knowledge Distillation Estándar:**
   El *student* aprende a replicar las distribuciones de probabilidad del *teacher* a nivel de tokens o palabras.
2. **TinyBERT:**
   Técnica que utiliza una combinación de destilación en múltiples etapas y optimización de capas intermedias.

3. **DistilGPT:**
   Reducción de un modelo GPT mediante la eliminación de capas y entrenamiento con *soft labels* generadas por un modelo GPT más grande.
4. **Sequence-Level Distillation:**
   El *student* se entrena para generar secuencias completas que se parezcan a las del *teacher* en tareas de generación de texto.
5. **Progressive Distillation:**
   Se entrena un *student* progresivamente con modelos intermedios cada vez más pequeños, permitiendo una transición gradual.
6. **Layer-wise Distillation:**
   El *student* se entrena para imitar los *hidden states* del *teacher* en cada capa correspondiente.
7. **Quantization-Aware Distillation:**
   Combinar destilación con técnicas de cuantización para reducir el tamaño del modelo aún más.